A Hybrid Linear Programming and Relaxed Plan Heuristic for Partial Satisfaction Planning Problems (extended abstract)*

J. Benton
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
j.benton@asu.edu

Menkes van den Briel
Dept. of Industrial Engineering
Arizona State University
Tempe, AZ 85287, USA
menkes@asu.edu

Subbarao Kambhampati
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
rao@asu.edu

Abstract

The availability of informed (but inadmissible) planning heuristics has enabled the development of highly scalable planning systems. Due to this success, a body of work has grown around modifying these heuristics to handle extensions to classical planning. Most recently, there has been an interest in addressing partial satisfaction planning problems, but existing heuristics fail to address the complex interactions that occur in these problems between action and goal selection. In this paper we provide a unique admissible heuristic based on linear programming that we use to solve a relaxed version of the partial satisfaction planning problem. We incorporate this heuristic in conjunction with a lookahead strategy in a branch and bound algorithm to optimally solve a class of over-subscriped planning problems.

Introduction

While some attempts have been made towards adapting relaxed plan heuristics (as used in FF (Hoffmann and Nebel 2001)) to PSP problems (Smith 2004; van den Briel *et al.* 2004), there is a fundamental mismatch. Relaxed plan heuristics are good at estimating the set of actions (and their cost) for achieving a given set of top level goals. In PSP problems, we do not up front know the goals that will be supported in the eventual optimal plan. The actions and the goals need to be selected together so as to optimize the net benefit. This requires a heuristic estimate (relaxation) with a more global "optimization" perspective.

A standard way of setting up a relaxation that is sensitive to global optimization perspective involves (i) setting up an integer programming (IP) encoding for the PSP problem and (ii) computing a linear programming (LP) relaxation of this encoding. In addition to being sensitive to the objectives of the optimization, such a relaxation is also sensitive to the negative interactions between the actions–something that is notoriously missing in the standard relaxed plan heuristics. One challenge in adopting this approach involves deciding on the exact type of IP encoding for the PSP problem. Although IP encodings for PSP problems have been proposed in the literature (Do *et al.* 2007), such encodings are made for bounded horizons. While this idea works for finding feasible plans, it does not work for finding optimal plans since it is not clear what step bound is required to guarantee optimality. In this paper, we adopt an encoding that is not dependent on the horizon bound.

One issue with the action encoding is that it gives the set of actions that could be part of the optimal plan but *does not* give the order in which they will occur. This is important because we would like to offset the cost of computing the heuristic by simulating the execution of the relaxed plan on the original planning problem using a lookahead method like that found in the planner YAHSP (Vidal 2004). As a partial remedy to this, we propose a novel way of combining the output of the linear programming relaxation of the action encoding with relaxed plan extraction. Specifically, the relaxed plan extraction is started with the goals "selected" by the LP solution, and is biased to choose actions that appear in the LP solution. [1]

We use the LP-relaxation of our encoding to provide heuristic values for each node in our search and incorporate it into a branch and bound search framework. Since all solutions are feasible in a PSP problem, the extracted relaxed plan is used to help find lower bounds (using lookahead), which allows the search to prune nodes that look unpromising. Because the LP provides informedness for negative interactions, the actions in our relaxed plan are more likely to mimic a solution to the original problem.

Problem Representation and Notation

Partial satisfaction planning with goal utility dependencies PSP^{UD} (Do *et al.* 2007) extends classical planning by assigning a utility value to sets of goals using k local utility functions, $f^u(G_k) \in \mathbb{R}$ on $G_k \subseteq G$, where any goal subset $G' \subseteq G$ has an evaluated utility value of $u(G') = \sum_{G_k \subseteq G'} f^u(G_k)$. This follows the *general additive independence* model for specifying utility (Bacchus and Grove 1995). In this way, any set of goals may be assigned a real valued utility. Additionally, each action a has an associated cost $cost(a)$, such that $cost(a) \geq 0$.

LP Heuristic

We present a unique admissible heuristic that solves a relaxation of the original PSP^{UD} problem by using the LP-

*An extended version of this paper will appear in the proceedings of ICAPS 2007.

[1] Given that the LP solution will be fractional, "selected by LP solution" is interpreted as goals (actions) that have values above a threshold.

relaxation of an IP formulation. While most heuristics ignore the delete effects of the actions, our heuristic accounts for the delete effects, but ignores action orderings instead. The formulation that we describe is based on the SAS+ planning formalism (Bäckström and Nebel 1995), where an SAS+ planning task is a tuple $\Pi = \langle V, A, s_0, s_* \rangle$ such that $V = \{v_1, \ldots, v_n\}$ represents a set of state variables, A is a finite set of actions, s_0 indicates the initial state and s_* denotes the goal variable assignments. Each $v \in V$ has a domain D_v and takes a single value f from it in each state s, stated as $s[v] = f$. Each action $a \in A$ includes a set of preconditions, $pre(a)$, post-conditions, $post(a)$, and prevail conditions, $prev(a)$.

IP Encoding

Unlike previous integer programming formulations ours does not use a step-based encoding. In a step-based encoding the idea is to set up a formulation for a given plan length and increment it if no solution can be found. Such an encoding may become impractically large, even for medium sized planning tasks.

The variables in our formulation indicate how many times an action is executed, and the constraints ensure that all the action pre- and post-conditions must be respected. The solution to our formulation provides a relaxation because it ignores action ordering. We call the heuristic h_{LP}.

The formulation requires as parameters: $cost(a)$, the cost of action a; $utility(v, f)$, the utility of achieving value f in variable v in the goal state; and $utility(k)$, the utility of achieving the goal utility dependency set G_k in the goal state. We also introduce the variables: $action(a) \in \mathbb{Z}^+$, the number of times action $a \in A$ is executed; $effect(a, v, e) \in \mathbb{Z}^+$, the number of times that effect e in state variable v is caused by action a; $prevail(a, v, f) \in \mathbb{Z}^+$, the number of times that the prevail condition f in state variable v is required by action a; $endvalue(v, f) \in \{0, 1\}$, a variable equal to 1 if value f in state variable v is achieved at the end of the solution plan (0 otherwise); $goaldep(k) \in \{0, 1\}$, a variable equal to 1 if goal dependency G_k is satisfied (0 otherwise).

The objective is to find a plan that maximizes the difference between the total utility that is accrued and the total cost that is incurred.

$$\text{MAX} \sum_{v \in V, f \in D_v} utility(v, f) endvalue(v, f)$$
$$+ \sum_{k \in K} utility(k) goaldep(k) - \sum_{a \in A} cost(a) action(a)$$

The constraints ensure that the action pre- and post-conditions are respected, and link the utility dependencies with their respective state variable values. We have the following types of constraints:

- Action implication constraints for each $a \in A$ and $v \in V$. The SAS+ formalism allows the pre-conditions of an action to be undefined (Bäckström and Nebel 1995). We model this by using a separate effect variable for each possible pre-condition that the effect may have in the state variable. We must, however, ensure that the number of times that an action is executed equals the number of effects and prevail conditions that the action imposes on each state variable.
- Effect implication constraints for each $v \in V$, $f \in D_v$. In order to execute an action effect its pre-condition must be satisfied.
- Prevail implication constraints for each $a \in A$, $v \in V$, $f \in D_v$. In order to execute an action prevail condition it must be satisfied.
- Goal dependency constraints for each goal dependency k. All values of the goal dependency are achieved at the end of the solution plan if and only if the goal dependency is satisfied.

We use the linear programming relaxation of this formulation as an admissible heuristic in our branch and bound framework.

Example: To illustrate the heuristic, let us consider a transportation problem where we must deliver a person, *per1* to a location, *loc2* using a plane, *p1*, and must end with the plan at *loc3*. The cost of flying from *loc1* to *loc2* is 150, from *loc1* to *loc3* is 100, from *loc3* to *loc2* is 200, and from *loc2* to *loc3* is 100. To keep the example simple, we start *per1* in the plane and the plane at *loc1*. There is a cost of 1 for dropping the person off. Having the person and plane at their respective destinations each give us a utility of 1000 (for a total of 2000).

The optimal plan for this problem is apparent. With a total cost of 251, we can fly from *loc1* to *loc2*, drop off *per1*, then fly to *loc3*. Recall that the LP heuristic, while it relaxes action ordering, works over SAS+ multi-valued fluents. The translation to SAS+ captures the fact that the plane, *p1*, can be assigned to only a single location. This is in contrast to planning graph based heuristics that ignore delete lists. Such heuristics consider the possibility that objects can exist in more than one location at a given step in the relaxed problem. Therefore, at the initial state, a planning graph based heuristic would return a plan that allowed the plane *p1* to fly from *loc1* to *loc2*, and *loc1* to *loc3*, putting it in multiple places at once.

In contrast, the solution from the LP-based heuristic for this problem at the initial state includes exactly every action in the optimal plan (though without an ordering). In fact, the value returned for these actions is "1.0". Though this is a small example, the behavior is indicative of the fact that the LP, through the encoding of multi-valued fluents, is aware that a plane cannot be wholey in more than one place at a time. In this case, the value returned (the *net benefit*, or $2000 - 251 = 1749$) gives us the perfect heuristic.

For the example problem simulating the execution of this plan would allow us to reach the goal optimally. But because the solution lacks no action ordering, we cannot expect to properly execute actions given to us by the LP. Additionally, the LP may return values other than "1.0" for actions. We must deal with cases where the LP returns non-integer values on the action variables and consider how to order the actions given to us.

Using a Planning Graph for Action Order

Using a lookahead technique like the one found in YAHSP (Vidal 2004) requires that we have a reasonable action ordering. However, our LP ignores action orderings and so we turn to planning graph based heuristics for their well-established virtue of giving non-cyclic causal relationships between actions. We exploit this and present a method of using the LP to guide relaxed plan extraction in a planning graph that is created by ignoring delete lists. This gives us a set of ordered actions that we may simulate in an effort to reach higher-quality states during search.

Recall that a relaxed planning graph is created by iteratively applying all possible applicable actions given the propositions available, thereby generating a union of the previously available propositions with the ones added by applying the actions. This can provide a cost estimate on reaching a particular proposition by summing the cost of each action applied to reach it, always keeping the minimum summed cost (i.e., we always keep the cheapest way to reach any proposition). After this, we can extract a relaxed plan from the planning graph by greedily finding the cheapest supporting actions for the set of goals.

In partial satisfaction planning we should only extract plans for sets of goals that appear to be beneficial (i.e., provide a high net benefit). We can use the LP for this, as it returns a choice of goals. Given that the LP can produce real number values on each variable (in this case a goal variable), we give a threshold, θ_G on their value. We have the final variable assignment for a goal g, $Value(g)$. If $Value(g) \geq \theta_G$ then we select g to be used in the plan extraction process.

The idea for extracting a relaxed plan using the LP as guidance is to prefer those actions that are selected in the LP. When extracting a relaxed plan, we first look at actions supporting propositions that are of the least propagated cost and part of the LP solution. If no such actions support the proposition, we default to the procedure of taking the action with the least propagated cost. Again, since the LP can produce fractional values for any variable we place a threshold on action selection, $theta_A$. If an action variable $Action(a)$, is greater than the threshold, $action(a) \geq \theta_A$, then that action is preferred in the relaxed plan extraction process given the described procedure.

Branch and Bound Search

A common method for solving combinatorial problems is to use a branch and bound search where a global bound is kept that represents the objective value of the best feasible solution found so far. In this type of search it is well known that quickly finding a good, close to optimal bound allows for more efficient pruning of the search space. An admissible heuristic is typically used at each search state, s, to determine the potential value that may be found by branching there. This type of branch and bound search is similar to a best-first search but it allows for situations where we want to continue searching despite having found a solution (even a locally optimal one). For the problem of finding the best (*maximum*) net benefit we use a branch and bound methodology. The algorithm necessitates that we keep a global *lower bound* that provides the value of the currently best-valued node (in terms of total *net benefit* or g-value). This allows us to maintain optimality despite choosing paths that may potenetially lead to in-optimal states.

The search algorithm combined with the heuristic focuses on quality in that it will not terminate unless an optimal solution is found. This attribute of branch and bound search is in contrast to the planner SPUDS (Do *et al.* 2007), whose inadmissible heuristic could cause it to stop searching without finding an optimal solution. Additionally, any solution that we are given can be checked against the bound found at the beginning of search (from the initial state).

Implementation

We created a planner called BBOP-LP (Branch and Bound Over-subscription Planning using Linear Programming, pronounced "bee-bop-a-loop") on top of the framework used for the planner SPUDS (Do *et al.* 2007), which is capable of solving the same type of planning problems and was written in Java 1.5. h_{LP} was implemented using CPLEX 10. In our experimental results, which are shown in the full paper, we found that its performance is overall better, in terms of the quality of the plans found given a time limit.

Future Work

An advantage to using LP-based heuristics is that they are malleable. We plan to add or change constraints in the LP encoding used for h_{LP} such that we can achieve better heuristic values. We also will explore new ways of using the LP with added constraints that give us orderings without the use of a planning graph. Further exploration will involve finding ways to encode PDDL3 temporal constraints.

References

F. Bacchus and A. Grove. Graphical model for preference and utility. In *Proceedings of the 11th Conference on Uncertainty in Artifi cial Intelligence*, pages 3–10, 1995.

C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

B. Bonet and H. Geffner. Heuristics for planning with penalties and rewards using compiled knowledge. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, 2006.

M.B. Do, J. Benton, M. van den Briel, and S. Kambhampati. Planning with goal utility dependencies. In *Proceedings of the 20th International Joint Conference on Artifi cial Intelligence (IJCAI 2007)*, pages 1872–1878, 2007.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artifi cial Intelligence Research*, 14:253–302, 2001.

D. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 393–401, 2004.

M. van den Briel, R. Sanchez, M.B. Do, and S. Kambhampati. Effective approaches for partial satisfaction (oversubscription) planning. In *Proceedings of the 19th National Conference on Artifi cial Intelligence (AAAI-2004)*, pages 562–569, 2004.

V. Vidal. A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 150–159, 2004.